Software Engineering for Mechatronics Applications

2017W1-MECH550C
2017W1-MECH575A

Instructor

Professor Arda Erol
Email : ardaerol@mail.ubc.ca
Classroom : ICICS X051 (both lecture and labs)
Time : Tuesdays & Thursdays from 11am to 12:30pm (lectures)
Tuesdays from 12:30 to 2:30pm (labs)

Learning Outcomes

By the end of this course, students will be expected to be able to write software applications in C#, running on various platforms, including embedded systems. They will be able to write computational applications and user interfaces, build networked applications, and will know the difference between real-time and non-realtime application programming. Emphasis will be on industry best practices, such as object oriented programming, early testing, and selection of development process model based on project type, team size, and team distribution. Having learned the common process models used in different corporations, students will be able to fit right in to a team in a wide range of corporations, whether they specialize in safety critical systems or consumer applications.

Course Schedule

Introduction to C#

Lecture topics:

- Why Software? Advantages/disadvantages. The (mostly) deterministic nature of software.
- What is C#?
- C# syntax basics
  - Case sensitivity
  - Whitespace insensitivity; code formatting (1 statement per line, indentation)
  - Comments
- Making the computer compute (Expressions)
- Assignment to a variable (what is a variable?)
- Basic data types (bool, byte, integer types, floating point types, decimal, char)

    - How data is stored in memory
    - Float vs. integer types
        - Integer division rounding
        - Integer overflow, checked/unchecked
        - NaN, +inf, -inf
    - Arrays: One dimensional, multidimensional (rectangular, jagged)
- Expressions revisited
    - Resultant data type
    - Operators: Arithmetic, relational (<,>, etc.), logical, bitwise, assignment (=, +=, <<=, etc.)
        - Order of evaluation
- Repeating without copy and paste (for, while, ~~do while~~)
- Making decisions (if/else, switch/case)
    - Loops revisited (break, continue)
- Packaging expressions and assignments into reusable groups (methods)
    - Arguments (copy, ref, out)
    - Return value
- Making it easier to pass data around (struct, class)
    - Value types vs. reference types
    - Boxing/unboxing
    - Nested types
- Grouping methods that work on the same data (and calling them classes)
    - Fields
    - Properties
    - Constructors, destructors, Finalize, Dispose
    - *this* & *base* keywords
    - Static data
    - Organizing classes into files (Partial classes)
    - Namespaces (using)
- Sharing common functionality (inheritance & polymorphism)
    - Abstract classes
    - Abstract and virtual methods, override
    - Access modifiers (public, protected, private, internal)
    - Sealed classes
    - Root of all classes (System.Object, object)
        - Equals, GetHashCode, GetType, ToString
    - Interfaces
- Strings (System.String, string)
- Enums
- Interfacing with the outside world
    - Methods that access hardware devices (screen, keyboard, disk, network card, etc.)
        - File I/O
    - Responding to events from hardware devices (interfaces, events, delegates)
- Exceptions (throw/catch)
- The Stack
    - Recursion
- The Heap
- Real-time applications
    - Real-time vs. high performance
    - Hard, firm, and soft real-time requirements

    - Levels of machine control (automatic control, CNC, etc.)
  - C# vs. C++
    - Garbage collection
  - Embedded systems
    - Embedded Linux (vs. other options)

Labs:

- Install Visual Studio
- Define a complex number type
- Calculate the absolute value of 1000 complex numbers
- Use the debugger
- Read input from a file; write results to another file

Object oriented design

Lecture topics

Note: Students will have already learned all the pillars of object-oriented design by this point, in the “Introduction to C#” section. In this section, we will walk through a practical example.

Turning it upside down: Breaking down a problem into its parts
- Representing a robot using objects

Lab

- Finalize our robot design by building a representation of its arms

Data structures

Lecture topics

- Arrays
- Inserting, removing, searching, and sorting
  - Classifying the performance and complexity of an operation (big-O notation)
    - Running time and space needs
- Linked Lists
  - Singly Linked Lists
  - Doubly Linked Lists
  - Circular Lists
- Stacks
- Queues
  - Priority queues
- Binary Trees
  - Balance problems

- Graphs (brief mention only)
- Hash Tables (Dictionaries)
- C#’s Generic Collections
  - List<T>, LinkedList<T>, Queue<T>, Stack<T>, HashSet<T>, SortedSet<T>, Dictionary<Key,Value>, SortedDictionary<Key,Value>, SortedList<Key,Value>, BitArray
  - foreach
- Relational Databases (brief mention only)
  - SQL
- C#’s LINQ and PLINQ (brief mentions only)

Lab

- Compare the performance of the different collection classes for:
  - Inserting an element
  - Searching for an element
  - Getting the next element

User interface design

Lecture topics

- Flavours of .NET
  - .NET Framework
  - .NET Core
  - Xamarin, Mono
  - .NET Standard 2.0 (future baseline for all .NET flavours)
  - .NET Native
- XAML
  - WPF
  - Universal Windows Platform (UWP)
  - Xamarin.Forms

Lab

- Build a simple user interface

Multi-threading

Lecture topics

- How to handle multiple things in parallel (even on a single core system)
  - Processes
  - Threads
- Sharing data among threads
  - Semaphores

        - How to protect shared resources against concurrent access
            - Race conditions
            - Avoiding busy waits
        - Producer-consumer problem (i.e. bounded-buffer access)
        - Deadlocks
    - Designing code to avoid/minimize sharing of data
- Job/message processing threads
    - Simplified resource sharing
    - Avoidance of thread creation overhead
- A common OS pattern
    - Main/user interface thread
    - Background threads
- Multithreading in C#
    - *Parallel* class (Parallel.For, Parallel.Foreach, Parallel.Invoke)
    - *Thread* class
    - *Task* class (brief mention)
    - Synchronization among threads within a single process
        - *lock* statement (and brief mentions of *Interlocked* and *Monitor* classes)
        - *Semaphore* class
        - *Barrier* class
    - Synchronization among threads of multiple processes
        - *Mutex* class*, etc.* (very brief mention only)
    - Timers

Lab

- Create two threads that read and process data from two different arrays, and keep a running sum in a shared variable
- Add inter-thread synchronization to prevent errors in the final total sum

Software engineering: specifications, design principles, agile development

---

Lecture topics

- Requirements analysis
    - Getting requirements from customer(s)
    - Speculative development
- Product specifications
    - Difference between requirements and specifications
    - Software specifications (textual, UML)
- Software design (textual, UML, code)
    - Software complexity, team size
- Software development process models
    - Waterfall model
        - Gantt charts
        - Unified Modeling Language (UML)

            - Class diagrams
            - Package diagrams
            - Statechart diagrams
            - Activity diagrams
            - Other diagram types (brief mention of each only)
            - Avoiding excessive use
    - Agile development
        - Scrum
    - Everything in between; how to decide on a process model
- Mind mapping

Lab

- Collect requirements for a product (several different product ideas, so no two students have the same product & team-type combination)
- Create a software specifications document
- Select a development process model (each student will be given fictional teams of different sizes and geographical distributions)
- Create a fictional timeline of your product development

Software testing and maintenance

Lecture topics

- Testing
    - Unit testing
    - Instrumenting
        - Profiling
        - Checking for memory leaks
    - Exploratory testing
    - Regression testing
    - As part of agile development
- Automating unit and regression tests
    - Nightly builds
- Role of an independent test team
    - Black-box testing
    - Requirements-based testing
        - Test cases
- Alpha and beta testing
- Version control
    - Team collaboration via version control (local branches, conflicts, etc.)
    - Distributed vs. centralized
- Defect tracking
- Release cycle, maintenance branches

Lab

- Collaborate with other students on a git repository
- Create an automated unit test
  - Introduce a bug to test your automated unit test
- Create releases on a git repository

Internet APIs and cloud computing

Lecture topics

- Internet Protocol (IP, IPv4 vs IPv6, NAT)
  - Domain Name System (DNS)
  - Datagrams vs. Streams (UDP, TCP)
    - Ports
  - HTTP/HTTPS
  - Standards Organizations (IETF, ISO/ANSI, W3C, ITU-T, IEEE, etc.)
- Data formats (HTML, XML, JSON, etc.)
- Deployment models
  - Client-server architecture
  - Peer-to-peer architecture
- Cloud computing (and storage)
  - Software as a service (SaaS)
  - Platform as a service (PaaS)
  - Infrastructure as a service (IaaS)
- Networking in C#
  - *Socket* class, *NetworkStream* class, *StreamReader*/*StreamWriter* classes, etc.

Lab

- Build an HTTP client to download data from the web
- Build a two-way chat system

Evaluation

- Attendance (10%)
- Assignments (lab-work) (50%)
- In-class test (10%)
- Final project (30%)

You will be assigned one final grade, which will then be given to the department as your final grade for both 2017W1-MECH550C and 2017W1-MECH575A (i.e. the same grade will be reported both).

Academic Misconduct

Make sure you understand UBC's Academic Misconduct policies, which can currently be found at the following link:

http://www.calendar.ubc.ca/Vancouver/index.cfm?tree=3,54,111,959

This course uses GitHub.ubc.ca for all assignments and projects. GitHub.ubc.ca uses your CWL to verify your identity. All work that is submitted using your CWL must be your own work without any exceptions. Protect your work and CWL password, as assisting other students by providing your work to them is also a serious form of academic misconduct. There are no exceptions to this policy in this course regardless of the nature of the assignment/project (collaborative* or not).

* All graded work (assignments/tests/projects) is non-collaborative unless explicitly identified by your instructor as collaborative in advance. For collaborative assignments/projects, **we (your instructor and TAs), not you,** may make your work available to other students automatically as you submit portions of it to GitHub.ubc.ca, in order to create an environment for collaboration. In such cases, we may set up teams of any size (anywhere from 2 people to the size of the entire class), and give the team shared access to a MECH550C-owned private repository on GitHub.ubc.ca without prior notice to you. You can check what teams you're on at GitHub.ubc.ca. Members of a team you're on can see any portion of your finished and unfinished work after you push the portion to a GitHub repository that is visible to the team, along with your commit comments for the portion and a detailed, full history of your changes.

Work that is not submitted to us does not need to follow the rules above. All work that is pushed to GitHub.ubc.ca is considered to be "submitted" work.

Your instructor reserves the right to ask you detailed questions about any of your assignment/project submissions and, if you cannot explain the inner workings of your own work, modify your already-assigned grade for the assignment/project before the term is over.

If you have any questions about whether a specific case would constitute plagiarism, contact your instructor in writing (e.g. by email) at least 48 hours before the assignment/project deadline, and wait for confirmation before submitting it.

Assignment deadlines

Assignments are due by 10:59 am on the Tuesday the week after the assignment is given to you. Any portion that is submitted at or after 11:00 am will not be accepted.

Final Project deadline

The final project is due by 11:59pm on Wednesday December 20th, 2017 for all teams. If your submission is delayed for any reason, 10 points (out of 100) will be deducted from your team score per day of delay for the first 4 days, and 20 points (out of 100) will be deducted for each additional day after that. Your individual score will be calculated based on the quality of your own code, your contribution to your team's success, and your team's score. The rules for this will be covered in class during the Nov 30 lecture, and posted to the MECH550C page on Blackboard Connect (connect.ubc.ca) after that.

GitHub errors:
In the event that you cannot push your finished/final work for an assignment/project to GitHub due to errors you can't resolve, you may send your final submission to your instructor via an email, so that you get a time-stamp to your submission before the assignment/project deadline. However, you must still resolve the GitHub issues as quickly as possible (via your TA's help if needed) and push the final version of your submission to GitHub.ubc.ca, so that the final version also exists on GitHub.ub.ca. Your version on GitHub must match the last copy you sent by email before the assignment/project deadline, unless the version on GitHub is newer and was successfully pushed before the deadline. Emailed submissions will never be marked or used for any purpose other than to verify the submission date of a delayed push to GitHub.